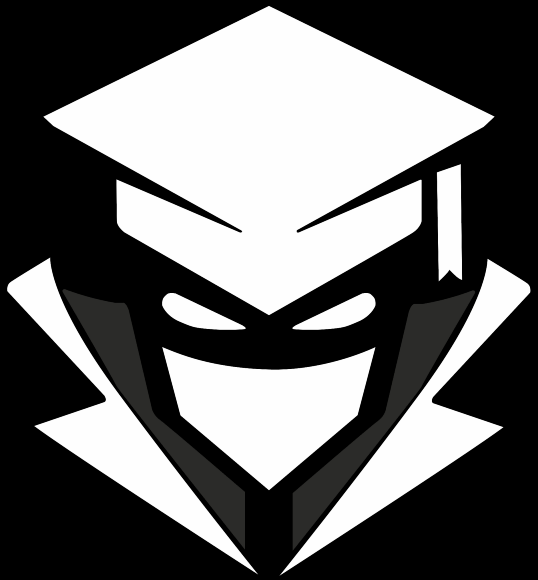


MISC

Cryptography

Sponsored By

accenture

WHAT IS CRYPTOGRAPHY?

Cryptography is the practice and study of techniques for secure communication in the presence of third parties.

GOALS OF CRYPTOGRAPHY

1. **Privacy**: Ensuring only intended recipients can read messages
2. **Authentication**: Proving one's identity
3. **Integrity**: Ensuring received messages have not been altered or corrupted
4. **Non-reupdiation**: Proving authorship of a message
5. **Key exchange**: Securely exchanging cryptographic keys

TYPES OF CRYPTOGRAPHY

- Asymmetric Crytography (public-key)
- Symmetric Cryptography (private-key)
- Hashing

SYMMETRIC CRYPTOGRAPHY

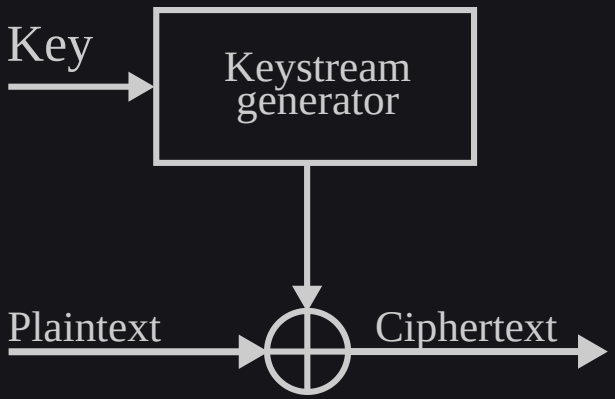
- Two parties use the same **shared key** for encryption and decryption
- The key needs to be kept **secret**
- Key is often exchanged using ideas from public-key cryptography (e.g. Diffie-Hellman key exchange)

Further classified into:

- Stream ciphers
- Block ciphers

STREAM CIPHERS

- Plaintext is combined with a **keystream**
- Keystream is usually taken from the output of random number generators
- Combination step typically involves **XORing** the plaintext with the keystream



ONE TIME PAD

- Uncrackable stream cipher
- Uses a key with the same length as the plaintext
- XORs each plaintext bit with a bit from the keystream
- ***A OTP key should only ever be used once***

$$m = 1011101011 \qquad k = 1110101101$$

$$\begin{aligned} E_k(m) = &\quad 1011101011 \\ &\oplus 1110101101 \\ = &\quad 0101000110 \end{aligned}$$

OTP IN PYTHON

Pure Python:

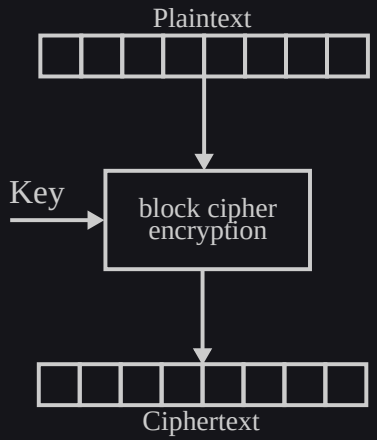
```
def xor(A, B):
    return bytes(a ^ b for a, b in zip(A, B))
```

Using pwntools:

```
from pwn import xor
xor(A, B)
```

BLOCK CIPHERS

- Operates on a fixed-length group of bits (called blocks)
- On its own, only suitable for encrypting a single block
- Different modes of operation allow block ciphers to be used more extensively



BLOCK CIPHERS (EXAMPLES)

- Data Encryption Standard (DES)
  - Introduced in early 1970s
  - No longer used
- Blowfish
  - Designed in 1993
  - Used in bcrypt
- Advanced Encryption Standard (AES)
  - First published in 1998
  - Supersedes DES
  - Current standard

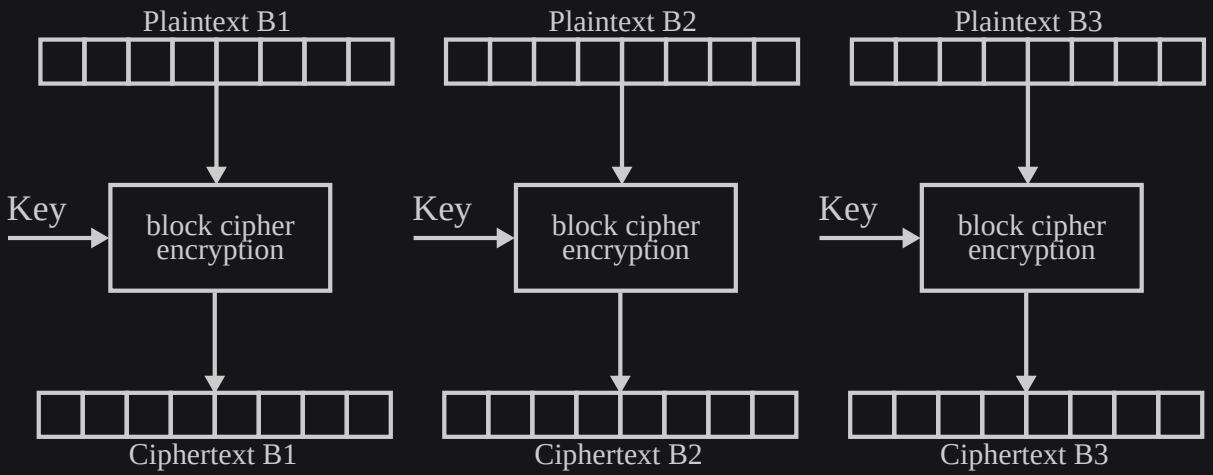
MODES OF OPERATION

- Allow block ciphers to be used on **arbitrary lengths** of data
  - Pads input plaintext and breaks them up into blocks
- Uses block ciphers - not actually ciphers themselves
- Most modes of operation use additional unique data (known as an initialisation vector (IV))
  - Ensures distinct ciphertexts
- Some common modes include:
  - **ECB**, **CBC**, CFB, OFB, CTR, GCM

ELECTRONIC CODEBOOK (ECB)

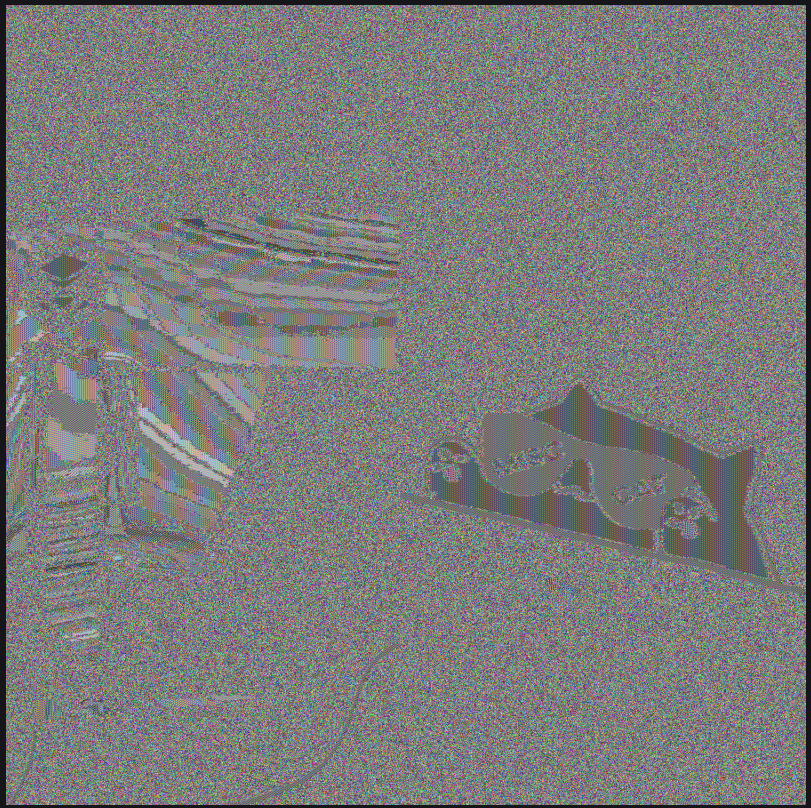
- Simplest mode of operation
- Plaintext is padded and then broken up into blocks
- Each block is **encrypted separately**
- Extremely broken - but makes for fun puzzles!

ELECTRONIC CODEBOOK (ECB)

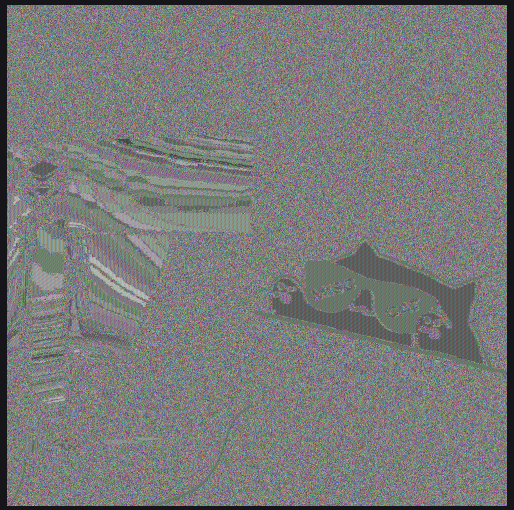


Each block is encrypted separately with the same encryption algorithm, using the same key... See where this might go wrong?

ELECTRONIC CODEBOOK (ECB)



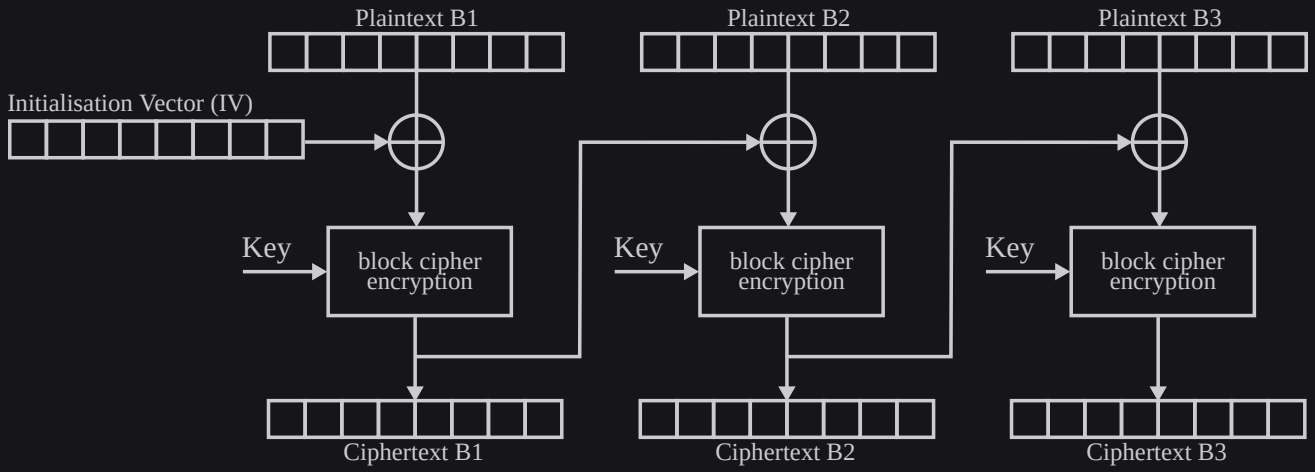
ELECTRONIC CODEBOOK (ECB)



CIPHER BLOCK CHAINING (CBC)

- Fixes the problem ECB had by **chaining** ciphertext blocks
- Each plaintext block is XORed with the **ciphertext of the previous block**
- First plaintext block is XORed with an initialisation vector (IV)
- The IV need not be secret, but **should not be reused**

CIPHER BLOCK CHAINING (CBC)



Identical blocks of plaintext will encrypt to different blocks of ciphertext

IV can be public, but should not be reused

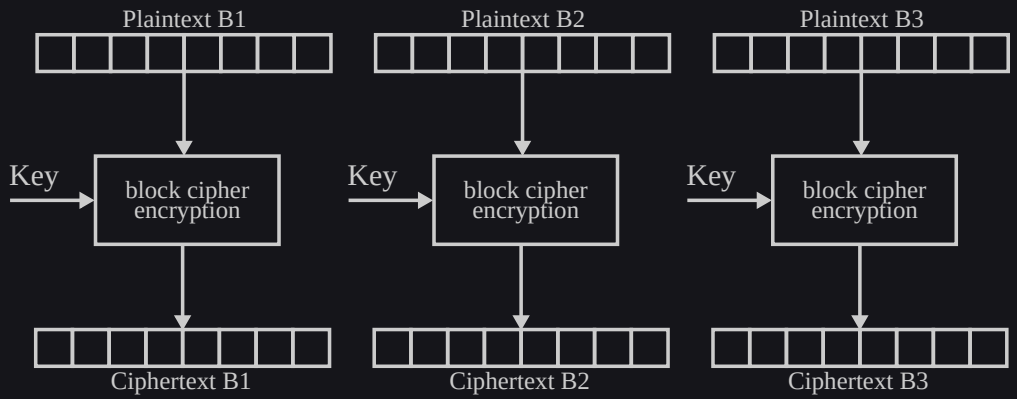
PUZZLE 1 - ECB ORACLE

Situation:

- Suppose you have access to an **ECB oracle** which takes an input you provide, **appends some secret text** to it, and sends you the encryption of that
- i.e. $\mathrm{ORACLE}(i) = \mathrm{ECB}_{\mathrm{k}}(i \,||\, \mathrm{secret})$
- You have **as many** encryption requests as you want
- There are no other constraints on what your input can be
- (Block size is 8 bytes)

Question: Can you determine the value of secret?

PUZZLE 1 - ECB ORACLE (HINTS)

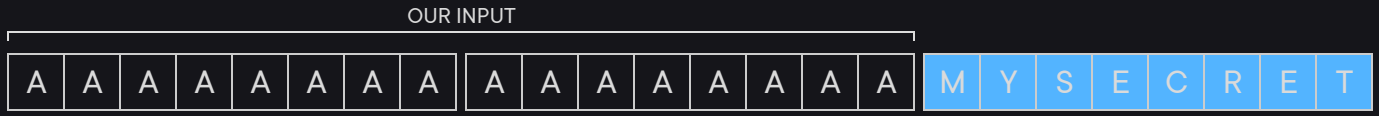


- Each plaintext block is encrypted separately
- Identical blocks of plaintext will have identical ciphertexts
- We can essentially ask for the encryption of any block of plaintext

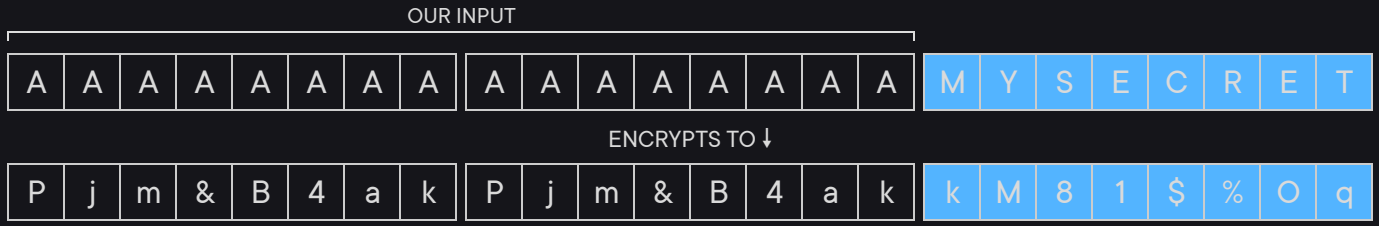
PUZZLE 1 - ECB ORACLE (HINTS)

- If we send $\underbrace{\mathrm{AAA}\ldots\mathrm{AAA}}_{16\ \mathrm{A's}}$, then blocks 1 and 2 of the ciphertext will be the same
- If we send $\underbrace{\mathrm{AA}\ldots\mathrm{AA}g_1\mathrm{AA}\ldots\mathrm{A}}_{7\ \mathrm{A's}\ ||\ g_1\ ||\ 7\ \mathrm{A's}}$, then blocks 1 and 2 of the ciphertext will be the same if $g_1 = s_1$ (where s_1 is the first byte of the secret)
- We can use this idea to bruteforce possible bytes of the secret, one at a time

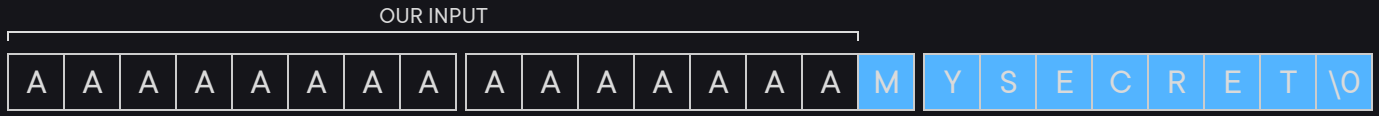
PUZZLE 1 - ECB ORACLE (HINTS)



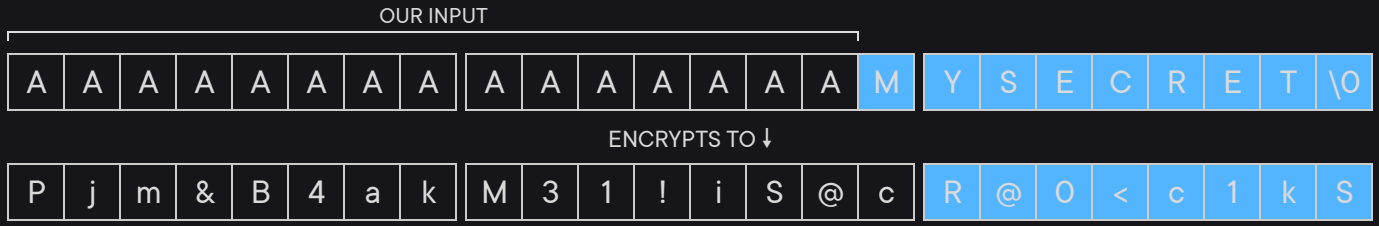
PUZZLE 1 - ECB ORACLE (HINTS)



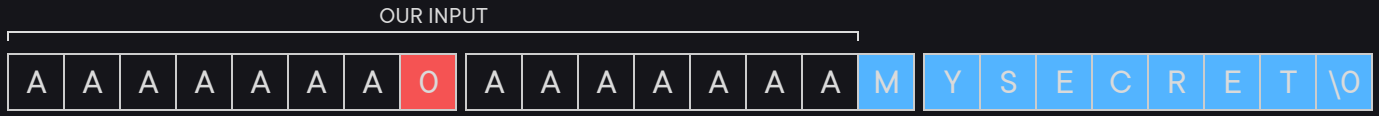
PUZZLE 1 - ECB ORACLE (HINTS)



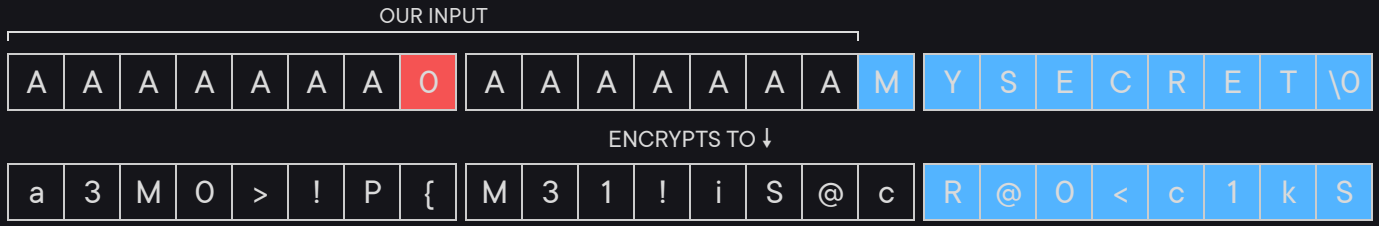
PUZZLE 1 - ECB ORACLE (HINTS)



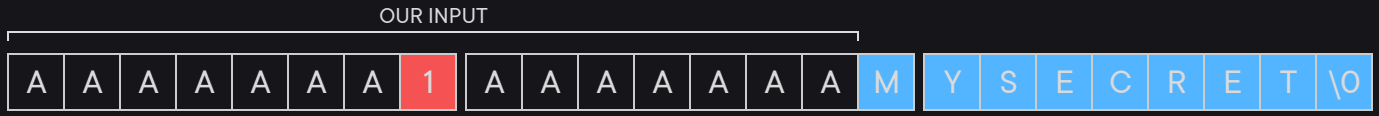
PUZZLE 1 - ECB ORACLE (HINTS)



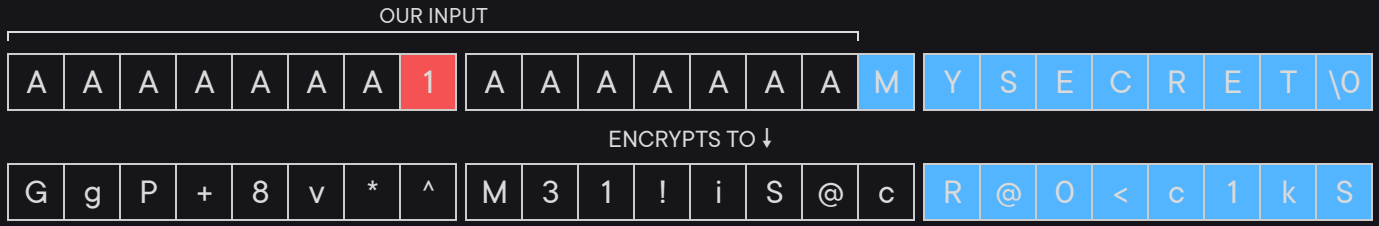
PUZZLE 1 - ECB ORACLE (HINTS)



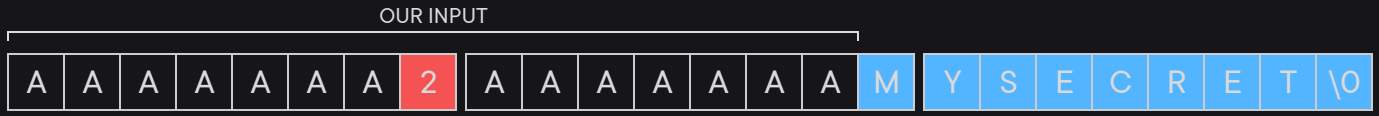
PUZZLE 1 - ECB ORACLE (HINTS)



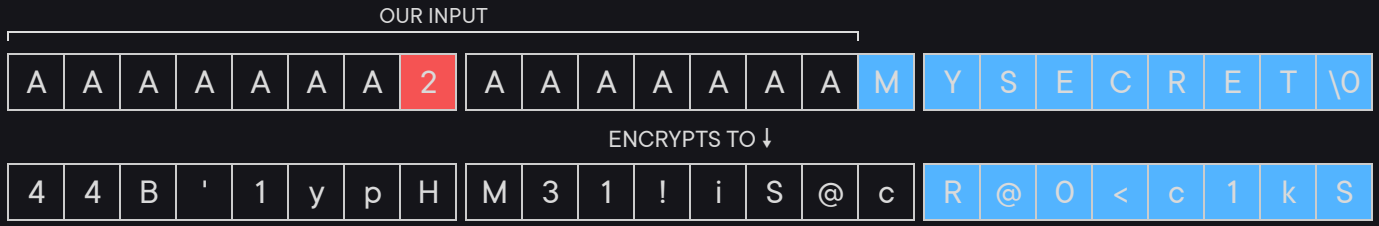
PUZZLE 1 - ECB ORACLE (HINTS)



PUZZLE 1 - ECB ORACLE (HINTS)



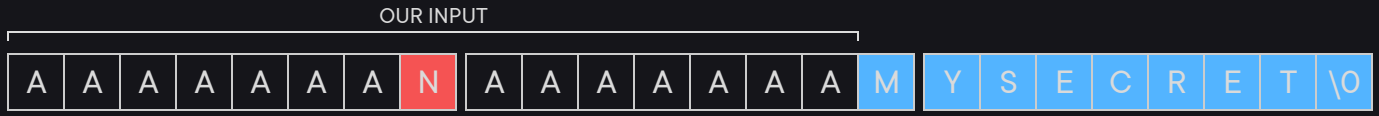
PUZZLE 1 - ECB ORACLE (HINTS)



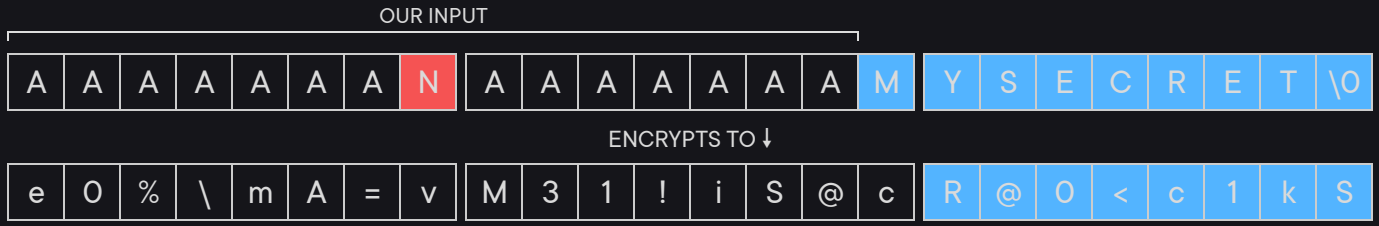
PUZZLE 1 - ECB ORACLE (HINTS)

Sooner or later...

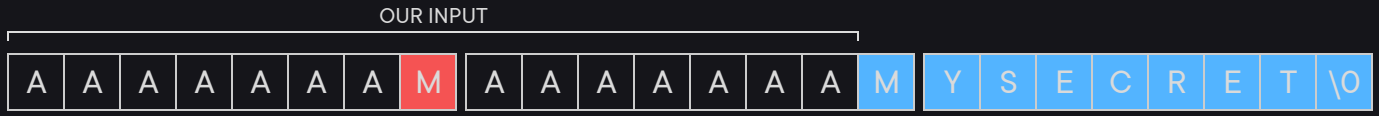
PUZZLE 1 - ECB ORACLE (HINTS)



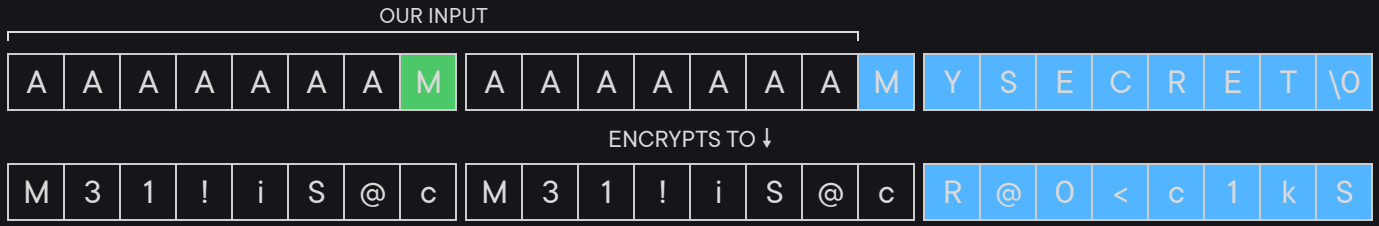
PUZZLE 1 - ECB ORACLE (HINTS)



PUZZLE 1 - ECB ORACLE (HINTS)

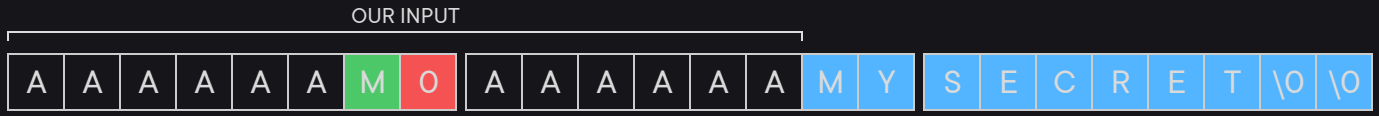


PUZZLE 1 - ECB ORACLE (HINTS)

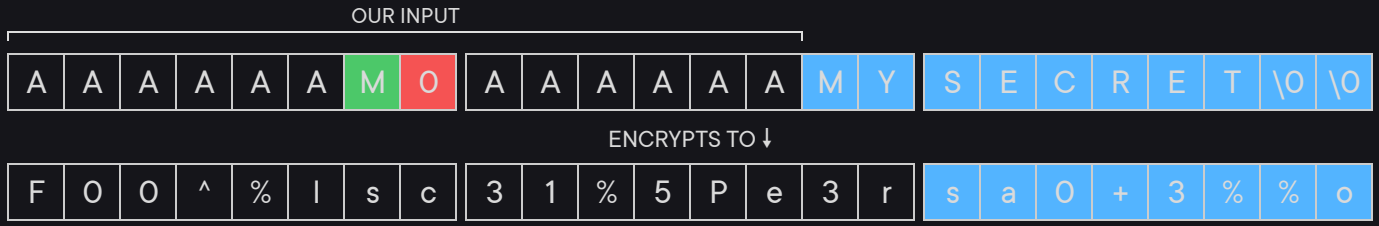


We found a match! Now we know that the first character of the secret text is `M`.

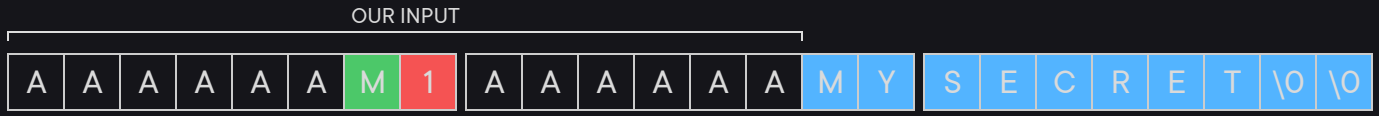
PUZZLE 1 - ECB ORACLE (HINTS)



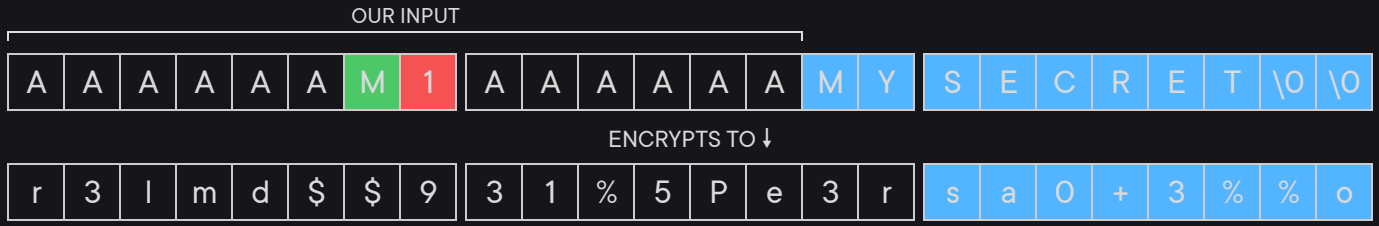
PUZZLE 1 - ECB ORACLE (HINTS)



PUZZLE 1 - ECB ORACLE (HINTS)



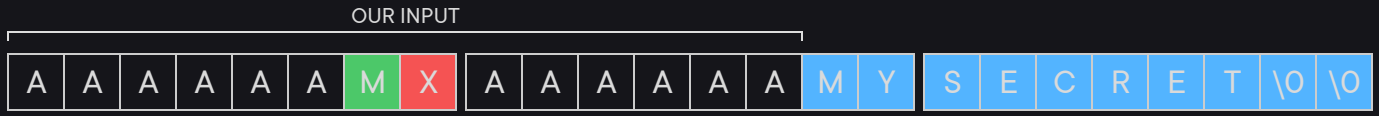
PUZZLE 1 - ECB ORACLE (HINTS)



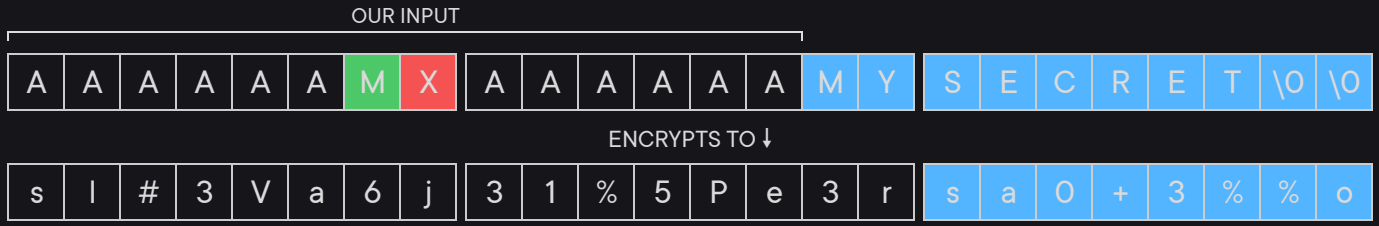
PUZZLE 1 - ECB ORACLE (HINTS)

And eventually...

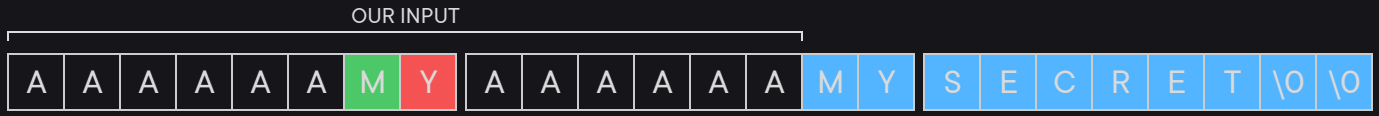
PUZZLE 1 - ECB ORACLE (HINTS)



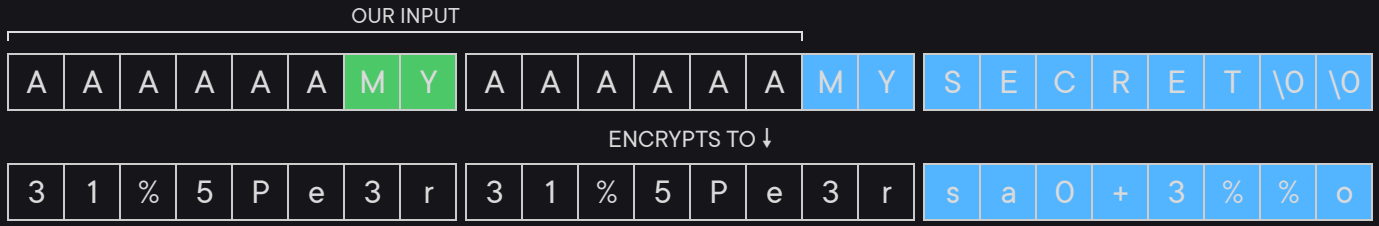
PUZZLE 1 - ECB ORACLE (HINTS)



PUZZLE 1 - ECB ORACLE (HINTS)



PUZZLE 1 - ECB ORACLE (HINTS)



We've found another match! If we continue this process, eventually we'll recover the entire secret...

PUZZLE 1 - ECB ORACLE (HINTS)

PUZZLE 1 - ECB ORACLE (HINTS)

PUZZLE 1 - ECB ORACLE (HINTS)

PUZZLE 1 - ECB ORACLE (HINTS)

We've fully recovered the secret!

PUZZLE 1 - ECB ORACLE

DEMO (maybe)

PUZZLE 1 - ECB ORACLE (ANALYSIS)

- How long did it take?
- If we used the naive approach, we would have had to try 256^n different inputs (!!)
- We've managed to reduce that to $256 \times n$

PUZZLE 2 - ECB/CBC

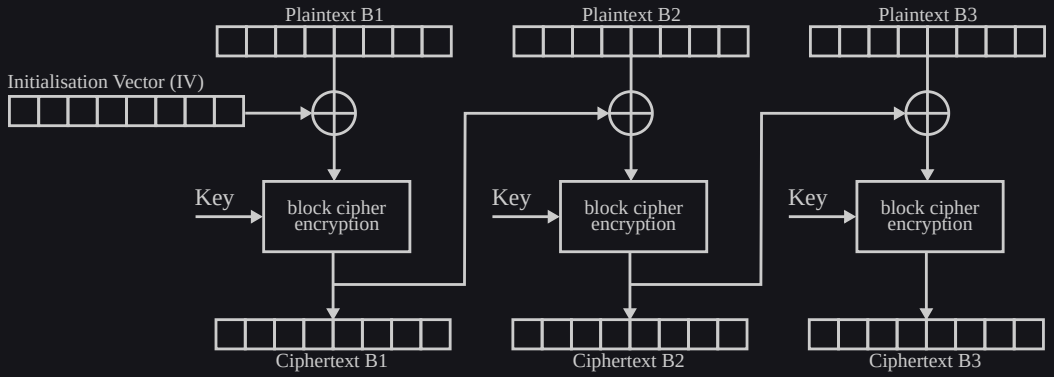
Situation:

- Suppose you have access to an **ECB decryption oracle** that decrypts anything you give it
- You are given some ciphertext that has been **encrypted using CBC** and the IV used
- The ECB decryption oracle and CBC encryptor use the same key
- You have **as many** decryption requests as you want

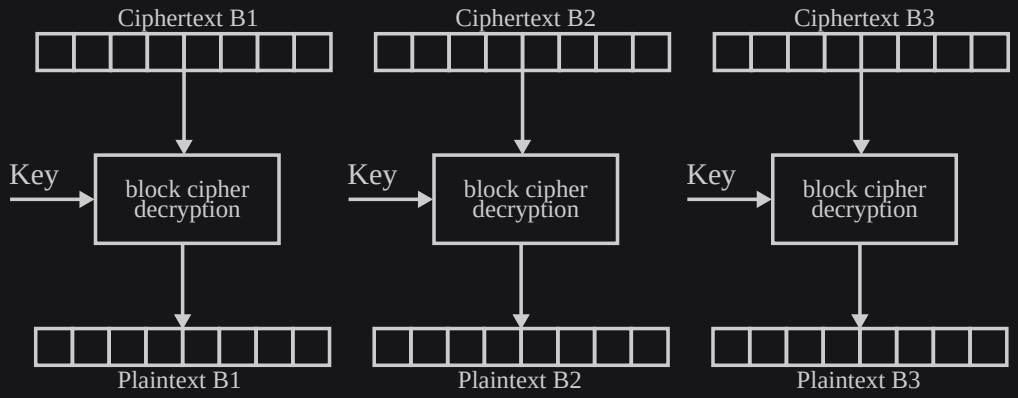
Question: Can you decrypt the ciphertext?

PUZZLE 2 - ECB/CBC (HINTS)

CBC Encryption:

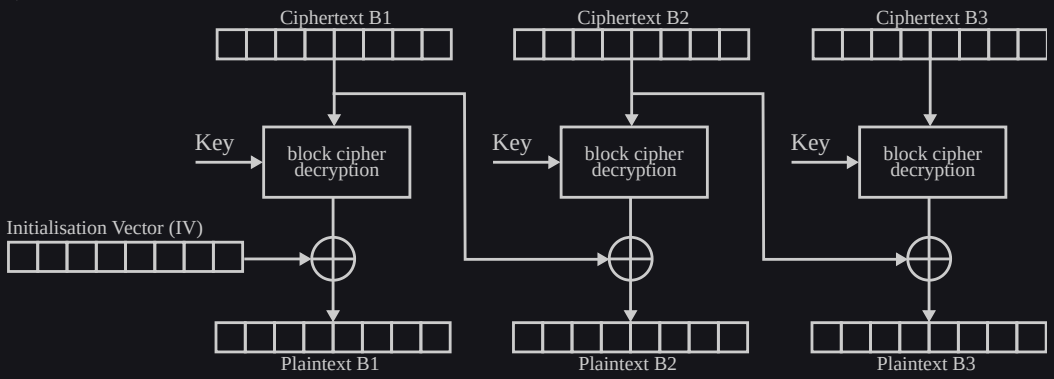


ECB Decryption:

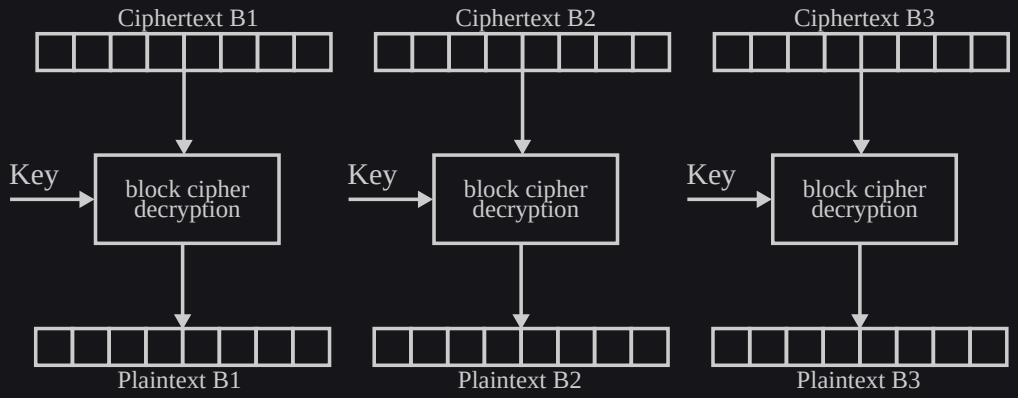


PUZZLE 2 - ECB/CBC (HINTS)

CBC Decryption:

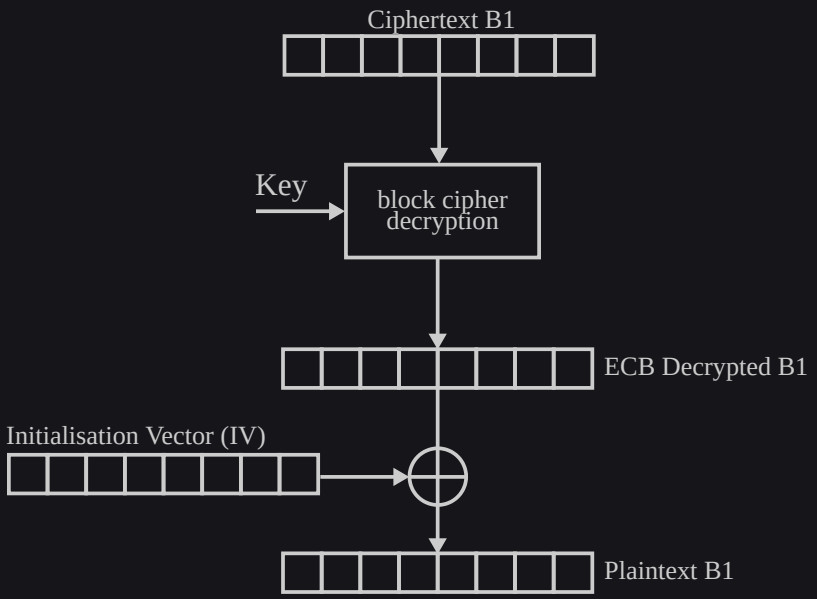


ECB Decryption:



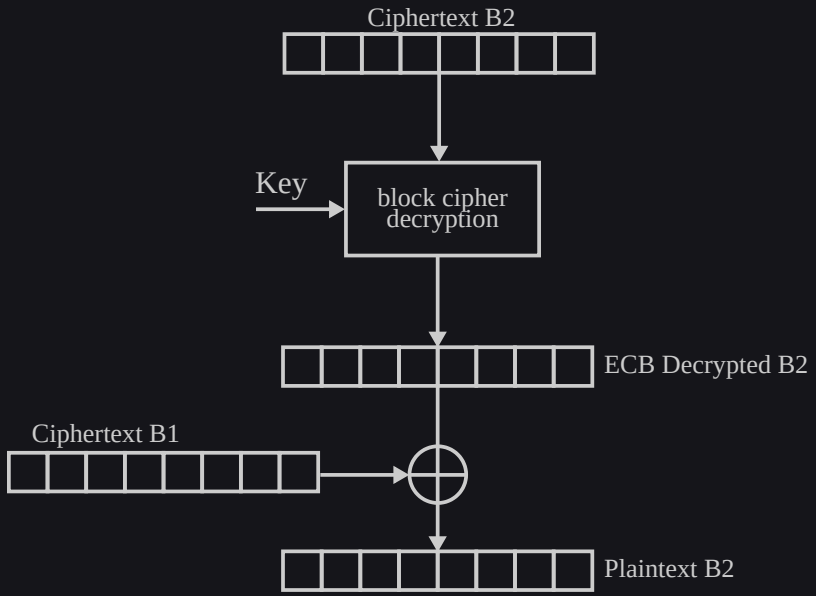
PUZZLE 2 - ECB/CBC (HINTS)

We can easily recover the first block of plaintext. We simply ask for the decryption of the first block of ciphertext, and XOR that with the IV.



PUZZLE 2 - ECB/CBC (HINTS)

Similarly for the second block, we XOR the output of the ECB decryption with the first block of ciphertext.



PUZZLE 2 - ECB/CBC (HINTS)

And so on, until we recover all blocks of the secret message...

PUZZLE 2 - ECB/CBC

DEMO (maybe)

PUZZLE 3 - WEAK KEYS

Situation:

- Suppose someone is using a small key (for whatever reason)
- They realise this could be a problem as their key might be bruteforced
- They decide to fix the problem by using another key to double the amount of security
- They encrypt their message first using the first key, and then encrypt the result using the second key
- You have access to *one* plaintext/ciphertext pair

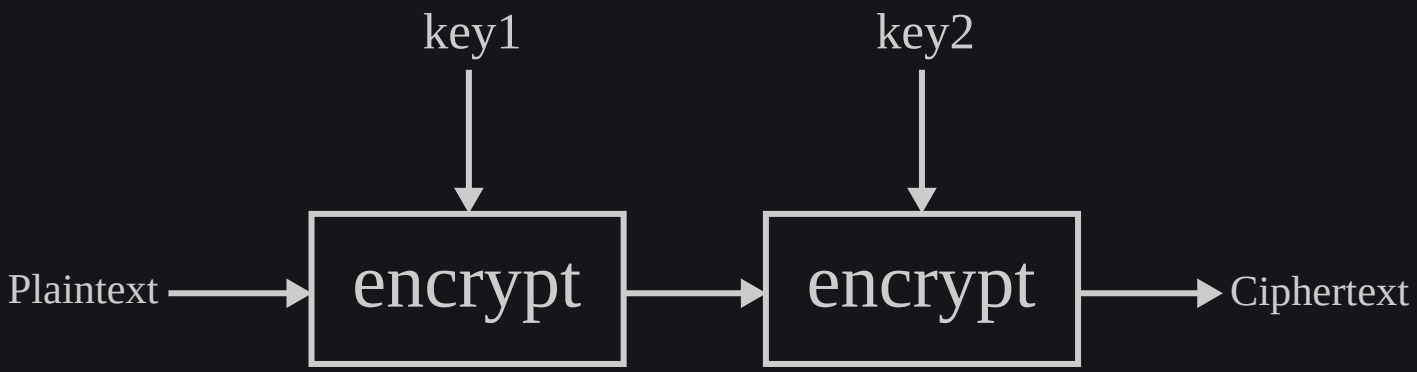
Question: Can you recover the keys?

PUZZLE 3 - WEAK KEYS

```python
import random
from Crypto.Cipher import AES
from string import printable

key1 = '0'*13 + ''.join(random.choice(printable) for _ in range(3))
key2 = ''.join(random.choice(printable) for _ in range(3)) + '0'*13

cipher1 = AES.new(key=key1, mode=AES.MODE_ECB)
cipher2 = AES.new(key=key2, mode=AES.MODE_ECB)

pt = "known plaintext!"

c1 = cipher1.encrypt(pt)
c2 = cipher2.encrypt(c1)
print('Plaintext string:', '"'+ pt + '"')
print('Encrypted string:', c2.hex(), '\n')

secret = "this is my very secret message!!"
s1 = cipher1.encrypt(secret)
s2 = cipher2.encrypt(s1)
print('Encrypted secret:', s2.hex())
```

```
Plaintext string: "known plaintext!"
Encrypted string: 78682dbb73be1eec02a66c3ad0172be7

Encrypted secret: 2e089837753a22092a5c002d311f9957bf060530242e2deaf691f063a64c
```

PUZZLE 3 - WEAK KEYS (HINTS)



PUZZLE 3 - WEAK KEYS (HINTS)

- Encrypting the plaintext with a key gives some value s_1
- Decrypting the ciphertext with a key gives some value s_2
- If $s1 = s2$, we've found the matching pair of keys

PUZZLE 3 - WEAK KEYS (HINTS)

- To bruteforce the keys, we don't necessarily need to bruteforce them together
- We can trade space for a faster algorithm
- Store the encryptions of the plaintext with a particular key in a lookup table
- Store the decryptions of the ciphertext with a particular key in a lookup table
- Check for keys which cause a collision
- i.e. find a pair of keys such that encrypting the plaintext with one key gives the same result as decrypting the ciphertext with the other

PUZZLE 3 - WEAK KEYS (HINTS)

This is known as a Meet-in-the-middle attack

PUZZLE 3 - WEAK KEYS

DEMO (maybe)

PUZZLE 3 - WEAK KEYS (ANALYSIS)

- Let n be the number of bits in one key (in our case, $n = 21$)
- Naive bruteforce approach: $O(2^n \times 2^n) = O(2^{2n})$ time and $O(1)$ space
- Meet-in-the-middle approach: $O(2^n)$ time and $O(2^n)$ space

FURTHER RESOURCES

- CryptoHack - Very good and fun challenges
- cryptopals - Self guided exercises
- Crypton - Readings + challenges
- Cryptography: An Introduction (Nigel Smart) - Introductory book on cryptography
- Introduction to Modern Cryptography (Katz & Lindell) - Yet another "introductory" book on cryptography

Slides: https://github.com/umisc/workshops